🚀 Production Deployment Guide

React + TypeScript | Node.js + Express | PostgreSQL (Supabase) | Socket.io | Hostinger

---

1. Overview

This document explains **how to deploy and manage the production environment** for your application. It is written to be usable by technical stakeholders, DevOps teams, or future developers, and can be referenced during go-live or future maintenance.

The application is a **full-stack system**, not a static website.

---

2. Production Architecture

Hosting Plan Clarification (Hostinger Cloud Startup)

The application is deployed using **Hostinger Cloud Startup** for the frontend. This plan is suitable and recommended for the UI layer, but it is important to understand its scope.

Cloud Startup supports:

- React frontend (production build)
- Static assets (JS, CSS, images)
- HTTPS / SSL
- CDN and high availability
- Git-based deployments or manual uploads

Cloud Startup does not support:

- Persistent Node.js servers
- Express APIs
- WebSockets / Socket.io
- Background workers or cron jobs

Because this application includes real-time features, chat, live dashboards, and Socket.io-based syncing, the backend must run separately from Cloud Startup. You'll need to purchase VPS for the server side – https://hpanel.hostinger.com/vps

---

High-level flow:

Browser (Users) → React Frontend (Hostinger) → Node.js / Express API + Socket.io (Hostinger VPS) → Supabase (Authentication + PostgreSQL)

Hosting Responsibilities

- **Frontend:** Hosted on Hostinger Web Hosting
- **Backend:** Hosted on Hostinger VPS (required for Node.js & WebSockets)
- **Database & Auth:** Supabase (managed PostgreSQL + authentication)
- **Real-time features:** Socket.io via backend server

---

3. Frontend Deployment (React + TypeScript)

Recommended Method: Git-Based Deployment

This is the cleanest and most reliable approach.

Steps

1. Push frontend code to a Git repository (GitHub recommended).
2. In Hostinger:
    - Connect the repository
    - Select the `main` branch
3. Configure build commands:
    - Install: `npm install`
    - Build: `npm run build`
4. Output directory:
    - Vite: `dist`
    - Create React App: `build`

Frontend Environment Variables (Production)

These values must be set in Hostinger before building:

- `VITE_SUPABASE_URL`

- `VITE_SUPABASE_ANON_KEY`
- `VITE_API_BASE_URL`
- `VITE_SOCKET_URL`

⚠️ **Never expose server-only secrets in the frontend** (database passwords, service role keys).

SPA Routing Configuration (Required)

Because this is a single-page application, all routes must fall back to `index.html`.

Add the following `.htaccess` file to the root directory:

```
None
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-f

RewriteCond %{REQUEST_FILENAME} !-d

RewriteRule . /index.html [L]
```

---

4. Backend Deployment (Node.js + Express + Socket.io)

Backend Hosting Requirement

While the frontend runs on **Hostinger Cloud Startup**, the backend must run on **a VPS or external Node hosting provider**. This is required to support:

- Persistent Node.js processes
- WebSockets (Socket.io)
- Real-time messaging and dashboards
- Background tasks and integrations

Recommended options:

- Hostinger VPS (preferred for same-provider simplicity)
- External providers such as Railway, Render, Fly.io, or DigitalOcean App Platform

The backend is typically exposed at:

- `https://api.yourdomain.com`

Hosting Requirement

The backend **must run on a VPS**. Shared hosting environments do not support:

- Persistent Node.js servers
- WebSockets (Socket.io)
- Background processes

VPS Setup (One-Time)

- OS: Ubuntu 20.04 or newer
- Install dependencies:

```
None
sudo apt update

sudo apt install nodejs npm

npm install -g pm2
```

Backend Environment Variables

Set the following on the VPS:

- `NODE_ENV=production`
- `PORT=3001`
- `SUPABASE_URL`
- `SUPABASE_SERVICE_ROLE_KEY`
- `DATABASE_URL`
- `JWT_SECRET`
- `CORS_ORIGIN=https://yourdomain.com`

Process Management (PM2)

Start and manage the backend using PM2:

```
None
pm2 start index.js --name app-backend

pm2 save

pm2 startup
```

PM2 ensures:

- Automatic restarts
- Crash recovery
- Server reboot persistence

Domain Routing & WebSockets

Recommended setup:

- Frontend: `https://yourdomain.com`
- Backend API: `https://api.yourdomain.com`

Use Nginx as a reverse proxy to support Socket.io:

```
None
location / {

  proxy_pass http://localhost:3001;

  proxy_http_version 1.1;

  proxy_set_header Upgrade $http_upgrade;

  proxy_set_header Connection "upgrade";

}
```

---

5. Supabase Configuration

Authentication Settings

In Supabase settings:

- **Site URL:**
  - `https://yourdomain.com`
- **Redirect URLs:**
  - `https://yourdomain.com/*`
  - `https://yourdomain.com/auth/callback`

OAuth providers must use matching URLs.

Database & Security

- PostgreSQL is fully managed by Supabase
- Backend connects using the **service role key**
- Frontend uses the **public anon key**
- Row Level Security (RLS) enforces role-based access

---

6. Real-Time Features (Socket.io)

Socket.io powers:

- Live dashboards
- Contests & leaderboards
- AI assistant messaging
- Real-time scorecards

Requirements:

- VPS hosting
- WebSocket upgrades enabled
- HTTPS across frontend and backend

---

7. Feature-Specific Notes

Permissions System

- Enforced at backend + database level
- Frontend reflects permissions but does not control access

Analytics & Exports

- Heavy queries executed server-side
- Export endpoints restricted by role

AI Assistant

- UI rendered client-side
- Messages processed via backend
- Socket.io maintains persistence across tabs

Integrations

- API tokens stored server-side only
- Sync status exposed to frontend

---

8. Go-Live Checklist

Frontend

- Production environment variables set
- SPA routing enabled
- HTTPS enforced

Backend

- Running on VPS
- PM2 active
- Nginx configured
- WebSockets functional

Supabase

- Site URL updated
- Redirect URLs verified

- RLS enabled

Testing

- Authentication flow
- Role-based access
- Real-time updates
- Dashboard data accuracy

---

9. Responsibility Split

Client / Hosting

- Maintains Hostinger account
- Owns domain and SSL
- Pays Supabase subscription

Application

- Codebase and deployments
- Backend server configuration
- Database schema and security
- Real-time infrastructure

---

10. Final Notes

This is a **production-grade SaaS architecture**. Attempting to run all components on shared hosting will cause failures in authentication, real-time features, and system stability.

A VPS + Supabase is the correct and supported setup.